WHAT IS CLAIMED IS:

1. A computer-implemented method for managing a plurality of processors as a virtual devices, said method comprising:

receiving a device request at a task queue manager running on a first processor in a computer system, wherein the computer system includes a plurality of heterogeneous processors that share a common memory and wherein the device request corresponds to a virtual device;

storing data corresponding to the request in the common memory;

identifying a second processor from the plurality of processors to handle the request;

signaling, from the first processor, the identified second processor;

receiving the data corresponding to the request at the second processor; and

processing the data by the second processor using software code stored in the second processor's local memory.

2. The method as described in claim 1 wherein the identifying further comprises:

detecting a utilization of one or more processors from the plurality of processors; and

selecting the second processor based upon its low utilization.

3. The method as described in claim 1 wherein the identifying further comprises:

detecting that the software code is loaded in the second processor's local memory.

4. The method as described in claim 1 further comprising:

creating, by the first processor, a task block in the common memory, the task block including a software code identifier; and

writing the address of the task block to a mailbox corresponding to the second processor.

5. The method as described in claim 4 further comprising:

receiving, at the second processor, the address of the task block from the second processor's mailbox;

retrieving, at the second processor, the software code identifier from the task block;

determining whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory:

reading the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

6. The method as described in claim 1 further comprising:

creating a plurality of queues, each of the queues corresponding to a virtual device;

writing the device request to a first queue from the plurality of queues, wherein the first queue corresponds to the virtual device; and

assigning the first queue to the second processor.

7. The method as described in claim 1 further comprising:

creating, by the first processor, a task block in the common memory, the task block including a software code identifier and an input buffer address; and

writing the address of the task block to a mailbox corresponding to the second processor.

8. The method as described in claim 7 further comprising:

receiving, at the second processor, the address of the task block from the second processor's mailbox;

retrieving, at the second processor, the software code identifier from the task block;

reading data from an input buffer located in the common memory at a location corresponding to the input buffer address into the second processor's local memory, wherein the reading is performed using a DMA operation;

determining whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory:

reading the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

9. The method as described in claim 1 further comprising:

receiving, at the first processor, a notification that the second processor has completed the processing; and

notifying, by the first processor, a software process from which the device request was received in response to receiving the notification from the second processor.

10. The method as described in claim 1 wherein the first processor and the second processor are dislike processors, wherein the first processor is adapted to execute a first instruction set and wherein the second processor is adapted to execute a second instruction set.

11. A computer-implemented method for using a processor as a virtual device, said method comprising:

initializing a task queue in a common memory, the task queue corresponding to a device;

receiving a device request from a first processor of a plurality processors in a computer system, wherein the plurality of processors share the common memory;

storing the request in the task queue and storing the input data corresponding to the request in the common memory;

assigning the task queue to a second processor selected from the plurality of processors;

signaling the second processor regarding the assignment;

loading the input data into a local memory corresponding to the second processor; and

processing the loaded input data using software code executing on the second processor.

12. The method as described in claim 11 wherein the assigning further comprises:

detecting a utilization of one or more processors from the plurality of processors; and

selecting the second processor based upon its low utilization.

13. The method as described in claim 11 wherein the assigning further comprises:

detecting that the software code is loaded in the second processor's local memory.

14. The method as described in claim 11 wherein the first processor and the second processor are dislike processors, wherein the first processor is adapted to execute a first instruction set and wherein the second

processor is adapted to execute a second instruction set.

15. The method as described in claim 11 further comprising:

creating, by the first processor, a task block in the common memory, the task block including a software code identifier; and

writing the address of the task block to a mailbox corresponding to the second processor.

16. The method as described in claim 15 further comprising:

receiving, at the second processor, the address of the task block from the second processor's mailbox;

retrieving, at the second processor, the software code identifier from the task block;

determining whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory:

reading the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

17. The method as described in claim 11 further comprising:

creating, by the first processor, a task block in the common memory, the task block including a software code identifier and the address of the input data; and

writing the address of the task block to a mailbox corresponding to the second processor.

18. The method as described in claim 17 further comprising:

receiving, at the second processor, the address of the task block from the second processor's mailbox; and

retrieving, at the second processor, the software code identifier and the address of the input data from the task block, wherein the loading includes reading the input data located at the input data address in the common memory into the second processor's local memory using a DMA operation.

19. The method as described in claim 11 further comprising:

receiving, at the first processor, a notification that the second processor has completed the processing; and

notifying, by the first processor, a software process from which the device request was received in response to receiving the notification from the second processor.

20. The method as described in claim 19 wherein the notifying further comprises:

writing completion data resulting from the processing to a data structure; and

providing the data structure to software process.

21. An information handling system comprising:

a plurality of heterogeneous processors;

a common memory shared by the plurality of heterogeneous processors;

a first processor selected from the plurality of processors that sends a request to a second processor, the second processor also being selected from the plurality of processors;

a local memory corresponding to the second processor;

a DMA controller associated with the second processor, the DMA controller adapted to transfer data between the common memory and the second processor's local memory; and

a virtual device tool for managing a plurality of processors as a virtual devices, the virtual device tool including software effective to:

receive a virtual device request at a task queue manager running on the first processor;

store data corresponding to the request in the common memory;

identify the second processor to handle the request from the plurality of processors;

signal, from the first processor, the identified second processor;

receive the data corresponding to the request at the second processor; and

process the data by the second processor using software code stored in the second processor's local memory.

22. The information handling system as described in claim 21 wherein the identification of the second processor further comprises software effective to:

detect a utilization of one or more processors from the plurality of processors; and

select the second processor based upon its low utilization.

23. The information handling system as described in claim 21 wherein the identification of the second processor further comprises software effective to:

detect that the software code is loaded in the second processor's local memory.

24. The information handling system as described in claim 21 further comprising software effective to:

create, by the first processor, a task block in the common memory, the task block including a software code identifier; and

write the address of the task block to a mailbox corresponding to the second processor.

25. The information handling system as described in claim 24 further comprising software effective to:

receive, at the second processor, the address of the task block from the second processor's mailbox;

retrieve, at the second processor, the software code identifier from the task block;

determine whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory, executing software effective to:

read the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

26. The information handling system as described in claim 21 further comprising software effective to:

create a plurality of queues, each of the queues corresponding to a virtual device;

write the device request to a first queue from the plurality of queues, wherein the first queue corresponds to the virtual device; and

assign the first queue to the second processor.

27. The information handling system as described in claim 21 further comprising software effective to:

create, by the first processor, a task block in the common memory, the task block including a software code identifier and an input buffer address; and

write the address of the task block to a mailbox corresponding to the second processor.

28. The information handling system as described in claim 27 further comprising software effective to:

receive, at the second processor, the address of the task block from the second processor's mailbox;

retrieve, at the second processor, the software code identifier from the task block;

read data from an input buffer located in the common memory at a location corresponding to the input buffer address into the second processor's local memory, wherein the reading is performed using a DMA operation;

determine whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory, perform software effective to:

read the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

29. The information handling system as described in claim 21 further comprising software effective to:

receive, at the first processor, a notification that the second processor has completed the processing; and

notify, by the first processor, a software process from which the device request was received in response to receiving the notification from the second processor.

30. The information handling system as described in claim 21 wherein the first processor and the second processor are dislike processors, wherein the first processor is adapted to execute a first instruction set and wherein the second processor is adapted to execute a second instruction set.

31. An information handling system comprising:

a plurality of heterogeneous processors;

a common memory shared by the plurality of heterogeneous processors;

a first processor selected from the plurality of processors that sends a request to a second processor, the second processor also being selected from the plurality of processors;

a local memory corresponding to the second processor;

a DMA controller associated with the second processor, the DMA controller adapted to transfer data between

the common memory and the second processor's local memory; and

a virtual device tool for managing a plurality of processors as a virtual devices, the virtual device tool including software effective to:

initialize a task queue in the common memory, the task queue corresponding to a virtual device;

receive a virtual device request from the first processor;

store the request in the task queue and store the input data corresponding to the request in the common memory;

assign the task queue to the second processor selected from the plurality of processors;

signal the second processor regarding the assignment;

load the input data into a local memory corresponding to the second processor; and

process the loaded input data using software code executing on the second processor.

32. The information handling system as described in claim 31 wherein the assignment further comprises software effective to:

detect a utilization of one or more processors from the plurality of processors; and

select the second processor based upon its low utilization.

33. The information handling system as described in claim 31 wherein the assignment further comprises software effective to:

detect that the software code is loaded in the second processor's local memory.

34. The information handling system as described in claim 31 wherein the first processor and the second processor are dislike processors, wherein the first processor is adapted to execute a first instruction set and wherein the second processor is adapted to execute a second instruction set.

35. The information handling system as described in claim 31 further comprising software effective to:

create, by the first processor, a task block in the common memory, the task block including a software code identifier; and

write the address of the task block to a mailbox corresponding to the second processor.

36. The information handling system as described in claim 35 further comprising software effective to:

receive, at the second processor, the address of the task block from the second processor's mailbox;

retrieve, at the second processor, the software code identifier from the task block;

determine whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory, perform software effective to:

read the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

37. The information handling system as described in claim 31 further comprising software effective to:

create, by the first processor, a task block in the common memory, the task block including a software code identifier and the address of the input data; and

write the address of the task block to a mailbox corresponding to the second processor.

38. The information handling system as described in claim 37 further comprising software effective to:

receive, at the second processor, the address of the task block from the second processor's mailbox; and

retrieve, at the second processor, the software code identifier and the address of the input data from the task block, wherein the loading includes reading the input data located at the input data address in the common memory into the second processor's local memory using a DMA operation.

39. The information handling system as described in claim 31 further comprising software effective to:

receive, at the first processor, a notification that the second processor has completed the processing; and

notify, by the first processor, a software process from which the device request was received in response to receiving the notification from the second processor.

40. The information handling system as described in claim 39 wherein the notification of the software process further comprises software effective to:

write completion data resulting from the processing to a data structure; and

provide the data structure to software process.

41. A computer program product stored on a computer operable media for managing a plurality of processors as a virtual devices, said computer program product comprising:

means for receiving a device request at a task queue manager running on a first processor in a computer system, wherein the computer system includes a plurality of heterogeneous processors that share a common memory and wherein the device request corresponds to a virtual device;

means for storing data corresponding to the request in the common memory;

means for identifying a second processor from the plurality of processors to handle the request;

means for signaling, from the first processor, the identified second processor;

means for receiving the data corresponding to the request at the second processor; and

means for processing the data by the second processor using software code stored in the second processor's local memory.

42. The computer program product as described in claim 41 wherein the means for identifying further comprises:

means for detecting a utilization of one or more processors from the plurality of processors; and

means for selecting the second processor based upon its low utilization.

43. The computer program product as described in claim 41 wherein the means for identifying further comprises:

means for detecting that the software code is loaded in the second processor's local memory.

44. The computer program product as described in claim 41 further comprising:

means for creating, by the first processor, a task block in the common memory, the task block including a software code identifier; and

means for writing the address of the task block to a mailbox corresponding to the second processor.

45. The computer program product as described in claim 44 further comprising:

means for receiving, at the second processor, the address of the task block from the second processor's mailbox;

means for retrieving, at the second processor, the software code identifier from the task block;

means for determining whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory:

means for reading the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

46. The computer program product as described in claim 41 further comprising:

means for creating a plurality of queues, each of the queues corresponding to a virtual device;

means for writing the device request to a first queue from the plurality of queues, wherein the first queue corresponds to the virtual device; and

means for assigning the first queue to the second processor.

47. The computer program product as described in claim 41 further comprising:

means for creating, by the first processor, a task block in the common memory, the task block including a software code identifier and an input buffer address; and

means for writing the address of the task block to a mailbox corresponding to the second processor.

48. The computer program product as described in claim 47 further comprising:

means for receiving, at the second processor, the address of the task block from the second processor's mailbox;

means for retrieving, at the second processor, the software code identifier from the task block;

means for reading data from an input buffer located in the common memory at a location corresponding to the input buffer address into the second processor's local memory, wherein the reading is performed using a DMA operation;

means for determining whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory:

means for reading the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

49. The computer program product as described in claim 41 further comprising:

means for receiving, at the first processor, a notification that the second processor has completed the processing; and

means for notifying, by the first processor, a software process from which the device request was received in response to receiving the notification from the second processor.

50. The computer program product as described in claim 41 wherein the first processor and the second processor are dislike processors, wherein the first processor is adapted to execute a first instruction set and wherein the second processor is adapted to execute a second instruction set.

51. A computer program product stored on a computer operable media for managing a plurality of processors as a virtual devices, said computer program product comprising:

means for initializing a task queue in a common memory, the task queue corresponding to a device;

means for receiving a device request from a first processor of a plurality processors in a computer

system, wherein the plurality of processors share the common memory;

means for storing the request in the task queue and storing the input data corresponding to the request in the common memory;

means for assigning the task queue to a second processor selected from the plurality of processors;

means for signaling the second processor regarding the assignment;

means for loading the input data into a local memory corresponding to the second processor; and

means for processing the loaded input data using software code executing on the second processor.

52. The computer program product as described in claim 51 wherein the means for assigning further comprises:

means for detecting a utilization of one or more processors from the plurality of processors; and

means for selecting the second processor based upon its low utilization.

53. The computer program product as described in claim 51 wherein the means for assigning further comprises:

means for detecting that the software code is loaded in the second processor's local memory.

54. The computer program product as described in claim 51 wherein the first processor and the second processor are dislike processors, wherein the first processor is

adapted to execute a first instruction set and wherein the second processor is adapted to execute a second instruction set.

55. The computer program product as described in claim 51 further comprising:

means for creating, by the first processor, a task block in the common memory, the task block including a software code identifier; and

means for writing the address of the task block to a mailbox corresponding to the second processor.

56. The computer program product as described in claim 55 further comprising:

means for receiving, at the second processor, the address of the task block from the second processor's mailbox;

means for retrieving, at the second processor, the software code identifier from the task block;

means for determining whether the software code corresponding to the software code identifier is loaded in the second processor's local memory; and

in response to determining that the software code corresponding to the software code identifier is not loaded in the second processor's local memory:

means for reading the software code from the common memory into the second processor's local memory, wherein the reading is performed using a DMA operation.

57. The computer program product as described in claim 51 further comprising:

means for creating, by the first processor, a task block in the common memory, the task block including a software code identifier and the address of the input data; and

means for writing the address of the task block to a mailbox corresponding to the second processor.

58. The computer program product as described in claim 57 further comprising:

means for receiving, at the second processor, the address of the task block from the second processor's mailbox; and

means for retrieving, at the second processor, the software code identifier and the address of the input data from the task block, wherein the loading includes reading the input data located at the input data address in the common memory into the second processor's local memory using a DMA operation.

59. The computer program product as described in claim 51 further comprising:

means for receiving, at the first processor, a notification that the second processor has completed the processing; and

means for notifying, by the first processor, a software process from which the device request was

received in response to receiving the notification from the second processor.

60. The computer program product as described in claim 59 wherein the means for notifying further comprises:

means for writing completion data resulting from the processing to a data structure; and

means for providing the data structure to software process.